

Z-Learning auf diskreten
Markov-Entscheidungsproblemen mit
zustandsabhängigen Aktionen

Tim Bossert

10. März. 2013

Bachelorarbeit

Matrikelnummer: 319710

Fakultät IV Elektrotechnik und Informatik
Institut für Softwaretechnik und Theoretische Informatik
Methoden der Künstlichen Intelligenz (KI)

Erstprüfer: Prof. Dr. Manfred Opper
Zweitprüfer: Prof. Dr. rer. nat. Klaus Obermayer
Betreuer: Dr. Andreas Rutor

Eidesstattliche Erklärung

Die selbständige und eigenhändige Ausfertigung versichert an Eides statt
Berlin, den 13. März. 2013

.....
Unterschrift

Inhaltsverzeichnis

1	Einleitung	3
1.1	Motivation	3
1.2	Rahmen der Arbeit	3
1.3	Aufbau	4
2	Reinforcement Learning	5
2.1	Klassische Markov Decision Problems	5
2.1.1	Reward	5
2.1.2	Die Markov Eigenschaft	6
2.1.3	Markov Decision Problem	8
2.1.4	Lösung eines Markov Decision Problem	8
2.1.5	Bewertungsfunktion	9
2.1.6	Bellman-Optimalitätsgleichung	10
2.1.7	Dynamische Programmierung	12
2.2	Q-Learning	13
2.2.1	Temporal-Difference Learning	13
2.2.2	Korrektheit vs. Effizienz	14
2.2.3	One-Step Q-Learning	15
2.3	Stetige Markov Decision Problems	16
2.3.1	Definition eines stetigen Markov Decision Problem	17
2.3.2	Erklärung zum neuen Modell	17
2.3.3	Lösung der angepassten Bellman-Optimalitätsgleichung	19
2.4	Z-Learning	21
2.4.1	Random Z-Learning	21
2.4.2	Verbesserte Episodengenerierung	22
2.5	Z-Learning lernt Aktionen	24
2.5.1	Z-Learning auf diskreten Markov Decision Problems	25
2.5.2	Überführung der unkontrollierten Dynamik \bar{P} in eine Strategie	25
2.5.3	Aktionen als reelle Vektoren in diskrete Aktionen übersetzen	25
3	Lernen des Übergangsmodell	27
3.1	Lernen von Aktionen in jedem Zustand	27
3.2	Lernen von Aktionen von Zustandandsgruppen	27

4	Setting	29
4.1	Ergebnisse	33
4.1.1	Fehlerbeschreibung	33
4.1.2	Lernen von Aktionen in jedem Zustand	34
4.1.3	Lernen von Aktionen von Zustandsgruppen	37
4.1.4	Vergleich Q- und Z-Learning	39
5	Zusammenfassung	41
5.1	Auswertung	41
5.1.1	Ausblick	42
5.1.2	Schluss	43

1 Einleitung

1.1 Motivation

Reinforcement Learning (dt. „Bestärkendes Lernen“) ist ein Teilgebiet des Maschinellen Lernen und der Künstlichen Intelligenz. Es beschäftigt sich mit dem Verhalten von Agenten in unbekanntem Umgebungen. Der Agent besitzt kein Model seiner Umgebung und muss daher durch Ausprobieren von Aktionen lernen welche Aktionen zu welcher Situation welchen Effekt haben. Der Vorteil dieses Ansatzes liegt in seiner Allgemeinheit und dadurch Anwendbarkeit auf viele Szenarien ohne das Model kennen zu müssen. So können Agenten nach einer Lernphase zum Beispiel Go spielen oder den Weg durch ein Labyrinth finden.

Was der Agent aber wissen muss ist die Menge von möglichen Aktionen von denen er wählen kann. Weiß er etwa nicht, dass er sich an einer Kreuzung im Labyrinth drehen kann, ohne zu wissen was *“Sich drehen“* bedeutet, so ist sehr wahrscheinlich, dass er niemals den Ausgang findet. Außerdem ist sich der Agent darüber im Klaren, dass seine Aktionen nicht immer gleich funktionieren. So kann das Fortbewegen fehlschlagen, wenn ein Stein im Weg liegt und der Agent ungünstig hinüberfährt. Die Aktionen sind also an Wahrscheinlichkeiten gebunden. Damit der Agent nun erfolgreich lernen kann, muss er wissen welche Aktionen zu welchem Zeitpunkt als vorteilhaft angesehen wird. Dies wird durch ein Feedback-System erreicht, schafft es der Agent zum Beispiel zum Ausgang des Labyrinths wird ihm ein positives Feedback gegeben, manövriert er sich hingegen in eine Sackgasse wird ihm dies als negativ signalisiert. Durch vielfaches Wiederholen dieses Lernprozess lernt der Agent eine Strategie die es ihm ermöglicht die ihm gestellte Aufgabe zu lösen.

1.2 Rahmen der Arbeit

Eine große Herausforderung ist es eine Lernmethode zu finden die es schafft dem Agenten eine optimale Strategie beizubringen, ohne jedoch zuviel Zeit zu benötigen. Diese Arbeit versucht zwei dieser Techniken vorzustellen, zu analysieren und zu vergleichen. Zum einem Q-Learning und ihr Nachfolger Z-Learning. Beide Ansätze stützen sich als grundlegendes Modell auf Markov Decision Problems (dt. „Markow-Entscheidungsproblem“) und auch diese werden in Grundzügen dargestellt. Insbesondere wird das Z-Learning betrach-

tet werden und mögliche Verbesserungen betrachtet und analysiert werden. Das grundlegende Problem ist, dass Z-Learning zwar schneller zur optimalen Strategie (siehe [3]) führt als Q-Learning, aber nur unter der Bedingung, dass Teile des Modells vorhanden sind. Damit Z-Learning auf der gleichen Informationsbasis wie Q-Learning arbeiten kann versuchen wir Ansätze zu beschreiben um diese Informationen aus der Umgebung zu extrahieren und somit den Nachteil vom Z-Learning zu neutralisieren. Der Preis dafür ist mehr Unsicherheit und mehr Aufwand als Z-Learning ohne zusätzliches Lernen.

1.3 Aufbau

Diese Arbeit ist in aufeinander aufbauende Kapitel geschrieben. Nach dieser Einleitung gibt es eine allgemeine Einführung in Reinforcement Learning. Darauf folgen Einführung in Reinforcement Learning, spezifisch Markov Decision Problems. Dann wird das Q-Learning in Zusammenhang mit diskreten Markov Decision Problems vorgestellt. Danach folgt eine Erklärung von stetigen Markov Decision Problems mit Z-Learning. Dann erfolgt die Vorstellung von Lösungen zur Entfernung einer Annahme im Z-Learning. Zuletzt erfolgt die Darstellung eines Settings in dem getestet wird, die Analyse der Ergebnisse und ein Zusammenfassung mit Ausblick.

2 Reinforcement Learning

Im folgendem werden die theoretischen Grundlagen des Reinforcement Learning vorgestellt. Sie sind notwendig zum Verständnis der Probleme und wie sie gelöst werden können. Behandelt werden diskrete und stetige Markov Decision Problems. Mit diesem Wissen werden dann Q- und Z-Learning eingeführt.

2.1 Klassische Markov Decision Problems

Der Term “Markov Decision Problem“ wurde bereits von Bellman in [2, 1] verwendet. Markov Decision Problems sind üblicherweise das Modell mit dem Reinforcement Learning Probleme formuliert werden. Ein Markov Decision Problem wird als sequentielles Entscheidungsproblem mit Markov Eigenschaft und akkumulierten Gewinn in einer vollständig beobachtbaren Umgebung beschrieben. Dieser Beschreibung folgt eine formale Definition und ihre Eigenschaften und Komponenten sollen im Weiteren beschrieben werden.

2.1.1 Reward

Das Ziel jedes Agenten beim Reinforcement Learning ist es den Reward zu maximieren. Immer wenn der Agent einen neuen Zustand erreicht erhält er vom System einen Reward. Dieser gibt dem Agenten Aufschluss darüber wie begehrenswert dieser Zustand ist.

Sei R der Reward und r_t der Reward für das Betreten eines Feldes zum Zeitpunkt t und N die Anzahl aller bisher ausgeführten Aktionen, dann gilt folgendes.

$$R = \sum_{i=t+1}^N r_t$$

Diese Definition ist nur sinnvoll unter der Annahme, dass der Agent nach endlich vielen Schritten einen Zielzustand erreicht, daher nennt man diese Art von Problemen auch episodisch. Bei Problemstellungen bei denen der Agent niemals terminiert muss ein Abzinsfaktor $0 \leq \gamma \leq 1$ hinzugefügt werden. Fügt man ihn nicht hinzu hat der Agent niemals den Ansporn sich zu verbessern, da sein Reward ins Unendliche hinauslaufen wird, was dazu

führt das der Agent niemals seine Aktionen evaluieren muss. Der Absinzfaktor muss auch mit jeder Iteration stärker werden, da er sonst nur einen skalierenden Charakter hat und keinen begrenzenden. Der Reward der dies berücksichtigt hat dann diese Form:

$$R = \sum_{i=0}^{\infty} \gamma^i r_{t+i+1}$$

Das Interessante am Absinzfaktor ist die Art und Weise wie er das Verhalten des Agenten beeinflussen kann. Wählt man einen Wert nahe der 1 versucht der Agent eine gute Langzeitstrategie zu finden - eine 1 ist äquivalent zu Rewards ohne Absinzfaktor. Wählt man jedoch einen Wert näher an der 0 werden die Rewards sehr schnell sehr klein und der Agent versucht eine Strategie zu finden die innerhalb der ersten Schritte möglichst hohe Rewards erhält, während die nachfolgenden Entscheidungen nicht mehr so einen großen Einfluss haben. Solche Arten von Problemen, bei denen es möglicherweise keinen Zielzustand gibt, nennt man kontinuierlich. Die Reihe konvergiert, es handelt sich um eine geometrische Reihe mit einem zusätzlichen Faktor. Dieser Faktor hat jedoch keinen Einfluss auf lange Sicht, da das geometrische Element stärker wirkt. Geometrische Reihen konvergieren immer wenn das geometrische Element $|\gamma| < 1$.

2.1.2 Die Markov Eigenschaft

Die Markov Eigenschaft, auch Gedächtnislosigkeit genannt, ist eine der wichtigsten Grundlagen von Markov Decision Problems und ist auch namensgebend.

In einem Reinforcement Learning Problem befindet sich der Agent zu jedem Zeitpunkt in einem Zustand mit bestimmten Charakteristiken, diese können vom Agenten wahrgenommen werden und beeinflussen die Aktionen, die der Agent zur Interaktion mit der Umgebung hat. Nach der Ausführung einer Aktion befindet sich der Agent in einem anderem Zustand und hat vom System einen Reward für seine Aktion erhalten. Die Aktionen die der Agent zur Auswahl hat sind an bestimmte Wahrscheinlichkeiten geknüpft und der neue Zustand, der durch das Ausführen einer gewählten Aktion erreicht wird, ist abhängig von diesen Wahrscheinlichkeiten.

Sind diese Wahrscheinlichkeiten, im folgenden Übergangswahrscheinlichkeiten genannt, nur abhängig von der gewählten Aktion und dem aktuellen Zustand

der Umgebung, spricht man von der Markov Eigenschaft. Es ist also nicht von Belang wie oft der Agent schon in diesem Zustand war oder welche Zustände vorher erreicht wurden.

Formal wird dies folgendermaßen beschrieben:

Sei z_t der aktuelle Zustand zum Zeitpunkt t . Sei a die gewählte Aktion. Sei r_t der Reward. Die Markov Eigenschaft ist gegeben, wenn folgendes gilt.

$$P(z_{t+1} = z', r_{t+1} = r | z_t, a_t, r_t, \dots, r_1, z_0, a_0) = P(z_{t+1} = z', r_{t+1} = r | z_t, a_t)$$

2.1.3 Markov Decision Problem

Mit diesem Verständnis können wir nun ein Markov Decision Problem definieren.

Ein endliches Markov Decision Problem ist ein 4-Tupel $(S, A_s, P_a(s, s'), R_a(s, s'))$

- S ist eine endliche Menge an Zuständen
- A_s ist eine endliche Menge von Aktionen im Zustand s , wobei $s \in S$
- $P_a(s, s') = P(s_{t+1} = s' | s_t = s, a_t = a)$ ist die Wahrscheinlichkeit, dass die Aktion a im Zustand s zum Zeitpunkt t in den Zustand s' zum Zeitpunkt $t + 1$ führen wird. Es ist das markovsche Übergangsmodell.
- $R_a(s, s')$ ist das Reward-Modell. Es beschreibt den Reward für das Erreichen des Zustands s' von Zustand s

Die Zustände und Aktionen müssen nicht zwingen endlich sein, die grundlegenden Algorithmen nehmen jedoch endliche Mengen an und vereinfachen zudem das Arbeiten mit diesen. Für einige unserer späteren Probleme sind endliche Mengen an Zuständen und Aktionen sogar zwingend. Eine weitere Annahme die wir treffen, neben der Markov Eigenschaft, ist, dass der Agent zu jedem Zeitpunkt weiß in welchem Zustand er ist. Das Zustandsmodell könnte auch mit einem probabilistischem Modell gekoppelt werden, sodass der Agent erst erschließen muss in welchem Zustand er ist.

2.1.4 Lösung eines Markov Decision Problem

Wie bereits beschrieben ist es das Ziel eines Agenten den Reward zu maximieren. Dies erreicht der Agent durch die Wahl der optimalen Aktion in einem bestimmten Zustand. Der Agent weiß jedoch nicht welche Aktion wann optimal ist und muss daher eine Strategie π_t entwickeln. Diese Strategie wird ständig weiterentwickelt und sollte nach jeder Aktion neu angepasst werden, daher der Index t . Formal sieht die Definition einer Strategie folgendermaßen aus:

Sei A die Menge aller Aktionen, sodass $A = \bigcup_{s \in S} A_s$

$$\pi_t : S \times A \rightarrow [0, 1]$$

$$\pi_t(s, a) \mapsto P(a_t = a | s_t = s)$$

Wobei gilt, dass $\sum_{a \in A} \pi_t(s, a) = 1$ und $a \notin A_s \Rightarrow \pi_t(s, a) = 0$

Die Strategie gibt an mit welcher Wahrscheinlichkeit, welche Aktion in welchem Zustand gewählt werden soll. Dabei sind nur Aktionen erlaubt die auch im aktuellen Zustand möglich sind. Es gibt hierbei zwei interessante Dinge zu beachten. Dieses allgemeine Schema betrachtet nicht die Rewards und erklärt auch nicht wie die Funktionswerte entstehen. Letzteres hängt von der gewählten Reinforcement-Learning-Technik ab. Zwei wichtige Techniken werden wir später beschreiben.

2.1.5 Bewertungsfunktion

Der Begriff Bewertungsfunktion ist stark mit dem Begriff der Strategie verknüpft. Üblicherweise ist die Grundlage von Reinforcement Learning Algorithmen eine Approximation der Bewertungsfunktion. Diese Bewertungsfunktion gibt an wie gut es für den Agenten ist sich in einem bestimmten Zustand zu befinden, bzw. wie vorteilhaft es ist welche Aktion in welchem Zustand auszuführen. Die Nähe zum Strategiebegriff insteht durch die Evaluierung der approximierten Bewertungsfunktion. Im Allgemeinen wird die Qualität der Bewertungsfunktion am Reward gemessen. Dieser Reward ergibt sich aus der Strategie π , welche der Agent verfolgt. Der erwartete Reward lässt sich daher formal folgendermaßen beschreiben:

$$V^\pi(s) = \mathbb{E}_\pi(R_t | s_t = s) = \mathbb{E}_\pi \left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right)$$

Es ist also der erwartete verminderte Reward, wenn sich der Agent im Zustand s befindet und die Strategie π verfolgt. In Anlehnung an diese Definition können wir die Güte einer Aktion a im Zustand s folgendermaßen definieren - unter der Annahme, dass der Agent sich an die Strategie π hält:

$$\begin{aligned} Q^\pi(s, a) &= \mathbb{E}_\pi(R_t | s_t = s, a_t = a) \\ &= \mathbb{E}_\pi \left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a \right) \end{aligned} \tag{1}$$

Das besondere und wichtige an Bewertungsfunktionen dieser Art ist die Rekursion, sodass die Bewertung des aktuellen Zustands s im Zusammenhang steht mit möglichen zukünftigen Nachfolgezuständen s' :

$$\begin{aligned}
V^\pi(s) &= \mathbb{E}_\pi(R_t | s_t = s) \\
&= \mathbb{E}_\pi \left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right) \\
&= \mathbb{E}_\pi \left(r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s \right) \\
&= \sum_{a \in A_s} \pi(s, a) \sum_{s' \in S} P_{ss'}^a \left[R_{ss'}^a + \gamma \mathbb{E}_\pi \left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_{t+1} = s' \right) \right] \\
&= \sum_{a \in A_s} \pi(s, a) \sum_{s' \in S} P_{ss'}^a [R_{ss'}^a + \gamma V^\pi(s')]
\end{aligned} \tag{2}$$

Diese Gleichung nennt man *Bellman-Gleichung* für V^π und gilt für alle π und s . Die Idee ist es alle möglichen Nachfolgezustände zu beachten, sodass der Wert eines Zustands sich aus dem Mittel der verminderten Werte der Nachfolgezustände, gewichtet nach ihren Wahrscheinlichkeiten, zusammen mit dem aktuellen zu erwartenden Reward ergibt.

2.1.6 Bellman-Optimalitätsgleichung

Wie bereits erwähnt, das Ziel eines Agenten ist es seinen Reward zu maximieren und dies erreicht er durch das Finden einer optimalen Strategie. Jede Strategie führt zu einer Bewertungsfunktion V^π . Auf diese Strategien kann man eine Ordnung \geq definieren, sodass gilt:

$$\pi \geq \pi' \Leftrightarrow \forall s \in S : V^\pi(s) \geq V^{\pi'}(s)$$

Die Qualität einer Strategie wird also gemessen an ihrem erwarteten Reward. Hat eine Strategie π einen größeren Reward als eine andere Strategie π' so sehen wir die Strategie π als besser, näher an der optimalen Strategie, an. Eine optimale Strategie π^* hat einen erwarteten Reward der mindestens so hoch ist wie der aller anderen Strategien. Formal ausgedrückt:

$$\forall \pi : \pi^* \geq \pi$$

Da eine optimale Strategie mindestens so gut ist wie alle andere Strategien ist, kann es daher mehrere optimale Strategien, jedoch haben alle die gleiche optimale Bewertungsfunktion.

$$\forall s \in S : V^*(s) = \max_{\pi} V^{\pi}(s)$$

$$\forall s \in S, a \in A_s : Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a)$$

Wie bereits gezeigt erfüllt V^* die Bellman-Gleichung. Da es sich um die optimale Bewertungsfunktion handelt, lässt sie sich ohne Bezug auf eine Strategie formulieren. Die daraus resultierende Gleichung hat den Namen *Bellman-Optimalitätsgleichung*

$$\begin{aligned}
V^*(s) &= \max_{a \in A_s} Q^{\pi^*}(s, a) \\
&= \max_{a \in A_s} \mathbb{E}_{\pi^*}(R_t | s_t = s, a_t = a) \\
&= \max_{a \in A_s} \mathbb{E}_{\pi^*} \left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a \right) \\
&= \max_{a \in A_s} \mathbb{E}_{\pi^*} \left(r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s, a_t = a \right) \\
&= \max_{a \in A_s} \mathbb{E}(r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a) \\
&= \max_{a \in A_s} \sum_{s' \in S} P_{ss'}^a [R_{ss'}^a + \gamma V^{\pi}(s')]
\end{aligned} \tag{3}$$

Analog dazu ergibt sich die Bellman-Optimalitätsgleichung für Q^*

$$\begin{aligned}
Q^*(s, a) &= \mathbb{E}_{\pi^*}(R_t | s_t = s, a_t = a) \\
&= \mathbb{E}_{\pi^*} \left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a \right) \\
&= \mathbb{E}_{\pi^*} \left(r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s, a_t = a \right) \\
&= \mathbb{E}(r_{t+1} + \gamma \max_{a' \in A_{s_{t+1}}} Q^*(s_{t+1}, a') | s_t = s, a_t = a) \\
&= \sum_{s' \in S} P_{ss'}^a [R_{ss'}^a + \gamma \max_{a' \in A_{s'}} Q^*(s', a')]
\end{aligned} \tag{4}$$

Diese Gleichung wurde von Richard Bellman entwickelt. Für endliche Markov Decision Problems besitzt die Bellman-Optimalitätsgleichung eine eindeutige Lösung V^* . Es handelt sich hierbei um ein Gleichungssystem mit

$n = |S|$ Gleichungen und n Variablen. Aufgrund des max-Operators sind die Gleichungen nicht linear. Sind die Rewards $R_{ss'}^a$ und die Übergangswahrscheinlichkeiten $P_{ss'}^a$ bekannt, kann das Gleichungssystem mit einem beliebigem Lösungsverfahren gelöst werden, sodass das optimale V^* einfach errechnet werden kann.

Wie bereits erwähnt gehen wir bei diesen Reinforcement Learning Aufgabenstellungen nicht davon aus, dass das gesamte Modell bekannt ist. Jedoch soll im Folgenden Kapitel kurz ein Verfahren der dynamischen Programmierung skizziert werden mit dem dieses Problem - unter der Annahme, dass das Modell bekannt ist - gelöst werden kann.

2.1.7 Dynamische Programmierung

Dynamische Programmierung ist eine Methode zum Lösen von komplexen Problemen indem das Problem in kleinere, einfachere und gleichartige Subprobleme aufgespalten wird. Die Ergebnisse der Subprobleme werden zwischengespeichert und wenn benötigt aufgerufen und zusammengesetzt. Dies nennt man Optimalitätsprinzip von Bellman. Die Bellman-Optimalitätsgleichung funktioniert auch nach diesem Prinzip und impliziert auch eine mögliche Update-Regel für einen iterativen Algorithmus. Dieser Algorithmus, genannt *Value Iteration*, fängt bei einer beliebigen Bewertung der Zustände $s \in S$ an und passt nach jedem Schritt die Bewertungen an.

$$\begin{aligned} V_{k+1}(s) &= \max_{a \in A_s} \mathbb{E}(r_{t+1} + \gamma V_k(s_{t+1}) | s_t = s, a_t = a) \\ &= \max_{a \in A_s} \sum_{s' \in S} P_{ss'}^a [R_{ss'}^a + \gamma V_k(s')] \end{aligned} \quad (5)$$

Für $k \rightarrow \infty$ konvergiert die Folge V_k gegen die optimale Strategie V^* , der Beweis dafür findet sich in [5]. Eines der Problem, welches die meisten unser Reinforcement Learning Algorithmen haben, ist das Terminieren. Da es das Ziel ist die optimale Strategie zu finden ist es nicht ersichtlich wann ein Algorithmus nahe genug an der Lösung ist. Daher benutzt man eine andere Größe als Indikator für die Nähe zur Lösung. Der Algorithmus wird terminiert, wenn die Differenz des neuen Wertes und des vorherigen Wertes kleiner ist als ein gegebener Wert δ . Da die initalen Werten bei der Value Iteration zufällig sind, wird manchmal auch eine Folge von n zu geringfügigen Änderungen erwartet. Es wird also abgebrochen, wenn der Algorithmus V nicht mehr in signifikanterweise verändert. Was signifikant ist muss je nach Problem neu

entscheiden werden.

2.2 Q-Learning

Im weiteren kehren wir nun zu unseren ursprünglichen Annahme zurück, dass das Modell unbekannt ist und damit das Reward-Modell und die Übergangswahrscheinlichkeiten. Somit können wir unseren vorherigen Ansatz nicht mehr verfolgen. Der Agent muss nun durch Interaktion mit der Umwelt die optimale Value Funktion approximieren. Der Agent folgt dem Schema: Zustand erkennen \rightarrow Aktion wählen \rightarrow Reward erhalten \rightarrow Zustand erkennen \rightarrow Aktion wählen \rightarrow Üblicherweise werden zum Lösen von Reinforcement Learning Problemstellungen entweder Monte Carlo Methoden oder Temporal-Difference Learning Methoden genutzt. Monte Carlo Methoden wählen in jedem Zustand eine zufällige Aktion aus und vergleichen im Zielzustand den Reward mit anderen zufällig erzeugten Aktionsfolgen, sodass nach sehr vielen Iterationen eine Approximation der Value Funktion vorliegt. Der andere Ansatz ist der des Temporal-Difference Learning bei dem nicht ein ganzer Satz von Aktionen betrachtet wird, sondern nach jeder Aktion neu evaluiert wird. Einer der bekanntesten Vertreter dieser Verfahren ist das Q-Learning. Die trivialste Variante ist das one-step Q-Learning, bei dem nach jeder Aktion die Action-Value-Approximation Q_t aktualisiert wird. Es gehört zu den bekanntesten Vertretern, da die Konvergenz zur optimalen Strategie zum ersten mal bewiesen werden konnte. Bevor wir jedoch direkt zum Q-Learning übergehen sollte noch ein paar Grundlagen des Temporal-Difference Learning erläutert werden und es wird kurz auf das Problem der Abwägung zwischen Exploration und Effizienz eingegangen.

2.2.1 Temporal-Difference Learning

Das Grundgerüst des Temporal-Difference Learning ist die Update-Regel der State-Value Funktion (dt. „Zustand-Wert Funktion“):

$$V(s_t) \leftarrow V(s_t) + \alpha_t[r_{t+1} + \gamma V(s_{t+1}) - V(s)]$$

Nach jeder durchgeführten Aktion, die aufgrund der Strategie π gewählt wurde, nimmt sich der Agent den Reward r_{t+1} den er durch das Betreten des neuen Zustands erhalten hat und bewertet den Zustand mit dem Reward und den bereits vorhandenen Bewertungen. α beschreibt die Lernrate des Agenten und nimmt über die Zeit ab, oftmals $\alpha_t = \frac{1}{t}$.

Jedoch haben wir noch keine Strategie mit der State-Value Funktion, denn diese gibt selbst keine Strategie an. Es werden keine Aktionen betrachtet sondern nur Zustände und weil das Modell nicht bekannt ist weiß der Agent nicht wie er von einem Zustand s in einen anderen Zustand s' kommen kann. Um dieses Problem zu umgehen wird deswegen die Action-Value Funktion (dt. „Aktion-Wert Funktion“) Q gelernt. Dadurch kann als Strategie einfach die bestbewertete Aktion für jeden Zustand gewählt werden, sodass gilt:

$$\pi(s) = \arg \max_{a \in A_s} Q(s, a)$$

2.2.2 Korrektheit vs. Effizienz

Zu Beginn eines jeden Lernprozess besitzt der Agent keinerlei Informationen über die Action-Value Funktion. Um die optimale Strategie zu finden wird der gesamte Zustandsraum nun mehrmals abgelaufen um diese zu finden. Das Problem dabei ist jedoch, dass dies oftmals sehr lange dauert und nicht unbedingt zielführend ist. So muss das Erkunden von zusätzlichen Wegen nicht zwingend zu einer besseren Strategie führen, da zum Beispiel eine optimale Strategie bereits gefunden wurde. Auf der anderen Seite ist es natürlich nicht erwünscht, dass sich der Agent nur auf bereits erkundeten Wegen bewegt und nur leicht vom besten bekannten Plan abweicht. Es könnte eine bessere Strategie geben, jedoch ist vielleicht ihr Anfang schlecht bewertet und der effizientere Teil wurde noch nicht erkundet. Der Agent sollte daher versuchen eine Balance zu finden zwischen dem Finden einer optimalen Strategie und der Zeit die er nutzt um eine hinreichend gute Strategie zu finden.

Folgt der Agent immer der bisher besten Strategie nennt man dies *greedy* und führt möglicherweise nicht zu einer optimalen Strategie. Einfache Abhilfe schafft hier die sogenannte ϵ -*greedy* Strategie. ϵ ist eine Zahl zwischen $[0, 1]$ und gibt an mit welcher Wahrscheinlichkeit die Aktion *greedy* gewählt werden soll, der Agent sich also an das bereits Bekannte hält oder der Agent eine zufällige Aktion zugunsten der Korrektheit wählen soll. Wählt man die Aktion komplett zufällig führt dies zu einer Gleichverteilung zwischen den Aktionen. Möchte man jedoch auch hier eine Beachtung der bisher geschätzten Werte, jedoch nicht immer die beste Aktion, gewichtet man jede Aktion entsprechend ihrer aktuellen Bewertung und wählt dann zufällig unter Beachtung der Gewichtung. Ein Beispiel dafür ist die Gibbs-Boltzmann-Verteilung:

$$\pi_t(s, a) = \frac{\exp \frac{Q_t(s, a)}{\tau}}{\sum_{a' \in A_s} \exp \frac{Q_t(s, a')}{\tau}}$$

Der Parameter τ bestimmt dabei wie *greedy* die Auswahl sein soll. Werte nahe 0 führen zu einer *greedy* Strategie, während größere Werte zu einer Gleichverteilung führen.

Das Problem der Wahl zwischen dem Finden der besten Strategie und dem Ausnutzen des bisherigen Wissen ist bereits bekannt unter dem Namen multi-armed bandit.

2.2.3 One-Step Q-Learning

Im folgenden soll nun das Q-Learning vorgestellt werden. Wie auch beim Temporal-Difference Learning beginnen wir mit der Update-Regel der Action-Value Funktion.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t [r_{t+1} + \gamma \max_{a \in A_{s_{t+1}}} Q(s_{t+1}, a) - Q(s_t, a_t)]$$

Dieser Algorithmus war der erste seiner Art und gilt als Durchbruch im Bereich des Reinforcement Learning. Es wurde zum ersten mal ein Algorithmus entwickelt der zur Evaluierung einer Aktion einer anderen Strategie folgen kann, als die Strategie die dafür verantwortlich ist zu entscheiden welche Aktion als nächstes ausgeführt werden soll. So ist es möglich zur Auswahl der nächsten Aktion einer ϵ -*greedy* Strategie zu folgen, während der Agent anhand einer einfachen *greedy* Strategie lernt. Zudem wurde auch bewiesen, dass dieser Algorithmus gegen die optimale Strategie konvergiert (siehe [9]). Eine notwendige Bedingung für das Finden dieser optimalen Strategie ist, dass alle Werte der Action-Value Funktion unendlich oft evaluiert werden. Ist dies gegeben konnte gezeigt werden, dass die Folge $Q_t(s, a)$ immer für $t \rightarrow \infty$ für alle $s \in S, a \in A$ gegen $Q^*(s, a)$ konvergiert, falls die Rewards beschränkt sind $|r_t| \leq r_{\max}$ und die Lernrate $0 \leq \alpha_t < 1$ folgende Bedingung erfüllt:

$$\sum_{i=1}^{\infty} \alpha_i = \infty, \quad \sum_{i=1}^{\infty} \alpha_i^2 < \infty$$

Für das Q-Learning können diese Konvergenzbedingungen sogar noch weiter abgeschwächt werden unter Zuhilfenahme der Theorie der stochastischen Approximation (siehe [8, 4]).

Das Q-Learning soll im Weiteren als Referenz angesehen werden für die Qualität anderer Reinforcement Learning Verfahren. Findet ein anderes Verfahren in weniger Iterationen eine hinreichend optimale Strategie, wird dieses Verfahren als besser angesehen. Im Folgenden zeigen wir eines dieser besseren Verfahren, doch müssen zuerst einige Grundlagen für stetige Markov Decision Problems beschrieben werden.

2.3 Stetige Markov Decision Problems

Als Erweiterung oder Alternative, je nach Interpretation, wurde zu klassischen Markov Decision Problems von Todorov [6] die Klasse der stetigen Markov Decision Problems eingeführt. Todorov konnte zeigen, dass stetige Markov Decision Problems die Lösung von Reinforcement Learning Problemen vereinfachen kann. Durch analytische Berechnung kann eine optimale Strategie errechnet werden und die Bellman-Optimalitätsgleichung kann in ein lineares Eigenwertproblem transformiert werden. Die Herleitung birgt zudem einen Ansatz für einen Algorithmus der ähnlich dem des Q-Learning funktioniert, bei dem die Strategie zur Auswertung und die Strategie zur Auswahl nicht zwingend gleich sein müssen.

2.3.1 Definition eines stetigen Markov Decision Problem

Ein stetiges Markov Decision Problem ist ein 4-Tupel $(S, U(i), P(u), l(i, u))$

- S ist eine endliche Menge an Zuständen
- $U(i)$ ist eine Familie zulässiger Aktionsmengen im entsprechenden Zustand $i \in S$
- $P(u)$ ist eine stochastische Matrix, deren Einträge $p_{ij}(u)$ die Übergangswahrscheinlichkeit von Zustand i nach Zustand j beschreibt
- $l(i, u)$ ist eine Abbildung, die beschreibt welche Kosten es hat die Aktion u zu wählen im Zustand i .

Eine zusätzliche Annahme, die für alle folgenden Probleme gelten soll, ist, dass es immer mindestens einen absorbierenden Zustand gibt und dieser auch erreicht werden kann. Man nennt dies indefinitem Horizont und es gilt: $\exists i \in S : \forall u \in U(i) : l(i, u) = 0$. Es kann auch mehrere absorbierende Zustände geben. Mit dieser Annahme ergibt sich die Bellman-Optimalitätsgleichung für die unverminderte, optimale State-Value Function v

$$v(i) = \min_{u \in U(i)} \left\{ l(i, u) + \sum_j p_{ij}(u)v(j) \right\}$$

Da in diesem Modell das Feedback nicht mehr als Reward, sondern als Kosten, aufgefasst wird, ist es nun das Ziel des Agenten die Kosten zu minimieren und was durch den *min*-Operator ausgedrückt wird.

2.3.2 Erklärung zum neuen Modell

Eine der Besonderheiten von Todorovs Modell ist die Modellierung der Aktionen. Der Aktionsraum ist der reelle Vektorraum $\mathbb{R}^{|S|}$. Eine Aktion wird durch einen reellen Vektor u beschrieben, dessen Dimension gleich der Anzahl aller Zustände des Markov Decision Problems ist. Dem Markov Decision Problem liegt eine unkontrollierte Markovkette zugrunde, daher ist es möglich die Übergangswahrscheinlichkeiten der Umwelt durch eine Matrix \bar{P} mit stochastischen Einträgen auszudrücken. Die Einträge \bar{p}_{ij} dieser Matrix beschreiben jene Übergangswahrscheinlichkeiten vom Zustand i in den

Zustand j . Die kontrollierte Übergangsmatrix $P(u)$ hat demnach folgende Form:

$$p_{ij}(u) = \bar{p}_{ij} \exp(u_j)$$

Daraus folgt $\bar{P} = P(0)$. Weiterhin muss jede Aktion $u \in U(i)$ die Bedingung erfüllen, dass die i -te Zeile $p_i(u)$ von $P(u)$ wieder eine diskrete Wahrscheinlichkeitsverteilung ist. Die Menge der zulässigen Aktionen im Zustand i ist daher:

$$U(i) = \left\{ u \in \mathbb{R}^{|\mathcal{S}|} \mid \sum_j \bar{p}_{ij} \exp(u_j) = 1 \right\}$$

Die Summe aller Übergangswahrscheinlichkeiten in einem Zustand j muss 1 sein damit es sich um eine zulässige Wahrscheinlichkeitsverteilung handelt. Zudem wirkt jede Aktion direkt auf das Übergangsmodell und verändert dieses. Die Kosten einer Aktion definieren wir daher als das Ausmaß der Veränderung der Übergangswahrscheinlichkeit, welche durch die Aktion ausgelöst wurde. Typischerweise benutzt man die Kullback-Leibler-Divergenz (KL-Divergenz) um dem Grad Veränderung zu messen. Die Aktionskosten sind daher:

$$r(i, u) = KL(p_i(u) || p_i(0)) = \sum_{j: \bar{p}_{ij} \neq 0} p_{ij}(u) \log \frac{p_{ij}(u)}{p_{ij}(0)}.$$

Die Eigenschaften der KL-Divergenz implizieren, dass $r(i, u) \geq 0$ und $r(i, u) = 0 \Leftrightarrow u = 0$. In Verbindung mit der Definition der kontrollierten Übergangsmatrix ergibt sich:

$$r(i, u) = \sum_j p_{ij}(u) u_j.$$

Die Kosten sind jedoch nicht nur die Kosten einer Aktion, das stetige Markov Decision Problem definiert selbst noch die Zustandskosten $q(i)$, diese müssen natürlich weiterhin auch mitbetrachtet werden. Die Gesamtkosten ergeben sich daher aus den Aktionskosten und den Zustandskosten $l(i, u) = r(i, u) + q(i)$. Die Bellman-Optimalitätsgleichung kann nun folgendermaßen dargestellt werden:

$$v(i) = \min_{u \in U(i)} \left\{ q(i) + \sum_j \bar{p}_{ij} \exp(u_j) (u_j + v(j)) \right\}$$

2.3.3 Lösung der angepassten Bellman-Optimalitätsgleichung

Mit dieser Bellman-Optimalitätsgleichung in angepasster Form kann nun eine analytische Methode zu ihrer Lösung angewandt werden. Zur Lösung dieses Extremalproblems können Lagrange-Multiplikatoren verwendet werden. Zuerst wird die Lagrange-Funktion gebildet:

$$\mathcal{L}(u, \lambda_i) = \sum_j \bar{p}_{ij} \exp(u_j)(u_j + v(j)) + \lambda_i \left(\sum_j \bar{p}_{ij} \exp(u_j) - 1 \right)$$

Die notwendige Bedingung für ein Extremum von \mathcal{L} bezüglich u_j ist

$$\frac{\partial \mathcal{L}}{\partial u_j} = \bar{p}_{ij} \exp(u_j)(u_j + v(j) + \lambda_i + 1) \stackrel{!}{=} 0$$

Falls $\bar{p} = 0$ ist $\bar{p}_{ij} = 0$ für alle u somit ohne Bedeutung für die Summe der Übergangswahrscheinlichkeiten. Falls $\bar{p} \neq 0$ ergibt sich die eindeutige Lösung:

$$u_j^*(i) = -v(j) - \lambda_i - 1$$

Für die zweite Ableitung von \mathcal{L} an der Stelle $u_j^*(i)$ gilt:

$$\frac{\partial^2 \mathcal{L}}{\partial u_j \partial u_j}(u_j^*(i)) = \bar{p}_{ij} \exp(u_j^*(i)) > 0$$

womit auch die hinreichende Bedingung erfüllt ist. Damit handelt es sich wirklich um ein Minimum. Werden nun der Nebenbedingung die Lagrange-Multiplikatoren durch die eindeutige Lösung ersetzt:

$$\begin{aligned} 1 &= \sum_j \bar{p}_{ij} \exp(-v(j) - \lambda_i - 1) \\ \Rightarrow \lambda_i &= \log \left(\sum_j \bar{p}_{ij} \exp(-v(j)) \right) - 1 \end{aligned}$$

Daraus ergibt sich die optimale Aktion:

$$u_j^*(i) = -v(j) \log \left(\sum_k \bar{p}_{ik} \exp(-v(k)) \right)$$

Damit wird die optimale Aktion in Abhängigkeit der optimalen State-Value Funktion beschrieben. Da letzteres bereits bekannt ist kann diese nun eingesetzt werden. Hierbei ist zu beachten, dass der min-Operator obsolet wird, da bereits der optimale Fall beschrieben wird.

$$\begin{aligned}
v(i) &= q(i) + \sum_j p_{ij}(u^*(i))(u_j^*(i) + v(j)) \\
&= q(i) + \sum_j p_{ij}(u^*(i))(-\lambda_i - 1) \\
&= q(i) - \lambda_i - 1 \\
&= q(i) - \log \left(\sum_j \bar{p}_{ij} \exp(-v(j)) \right) \\
&\Rightarrow \\
\exp(-v(j)) &= \exp(-q(i)) \sum_j \bar{p}_{ij} \exp(-v(j))
\end{aligned} \tag{6}$$

Todorov führt nun eine exponentielle Transformation

$$z(i) := \exp(-v(i))$$

ein, sodass die Bellman-Optimalitätsgleichung linearisiert wird:

$$z(i) = \exp(-q(i)) \sum_j \bar{p}_{ij} z(j)$$

Nun wird diese Gleichung in eine Matrixschreibweise überführt. Dafür wird ein Vektor z definiert, dessen Einträge $z(i)$ sind, und eine Diagonalmatrix $G = \text{diag}(\exp(-q(i)))$. Dies führt zu folgendem Eigenwertproblem:

$$z = G\bar{P}z$$

Für die Lösung ist nun ein Eigenvektor z von $G\bar{P}$ zum Eigenwert 1 gesucht. Zudem muss gelten:

$$\forall s \in S : z(s) > 0$$

$$s \in A \Rightarrow z(s) = 1$$

Ein Vektor, der diese Bedingungen erfüllt, existiert, da die von uns angepasste Bellman-Optimalitätsgleichung derartig konstruiert wurde. Eine Möglichkeit

diese Lösung zu finden wurde von Todorov selbst vorgeschlagen. Die Grundidee ist die einer iterativen Potenzmethode. Ein möglicher Ansatz ist:

$$z_{k+1} = G\bar{P}_{z_k}, \quad z_0 = 1.$$

Dies konvergiert gegen den Eigenvektor zum größten Eigenwert, welcher 1, denn \bar{P} ist eine stochastische Matrix und hat den Spektralradius $\rho(\bar{P}) = 1$ und die Diagonaleinträge $\exp(-q(i)) \leq 1$ der Matrix G skalieren das Spektrum runter. Da die Bellman-Optimalitätsgleichung eine eindeutige Lösung hat, gilt $\rho(G\bar{P}) = 1$ und z ist der Eigenvektor zum Eigenwert 1.

Sind die $z(i)$ bestimmt, können die Werte der State-Value Funktionen errechnet werden, indem die Umkehrfunktion $v(i) = -\log(z(i))$, der von Todorov eingeführten Transformation.

Damit kann nun die State-Value Funktion durch Iteration berechnet werden. Dieses Verfahren wird Z-Iteration genannt.

2.4 Z-Learning

Ähnlich der Value Iteration wurde im letzten Abschnitt ein Verfahren vorgestellt, welches in der Lage ist ein stetiges Markov Decision Problem zu lösen. Diese Lösung beinhaltet das Lösen eines Eigenwertproblems einer minimierten Bellman-Optimalitätsgleichung. Dies ist jedoch nur möglich wenn vollständiges Wissen über das Modell vorliegt.

Im Allgemeinen ist dies jedoch nicht der Fall und der Agent muss dieses Modell durch Interaktion mit der Umwelt approximieren, beziehungsweise lernen.

2.4.1 Random Z-Learning

Glücklicherweise liegt in dem von Todorov konstruierten Modell von stetigen Markov Decision Problems ein Ansatz für einen Algorithmus zur stochastischen Approximation einer optimalen State-Value Funktion vor. Dieser Algorithmus erlaubt, ähnlich dem Q-Learning, dass die Strategie zur Auswahl einer Aktion nicht zwingend die Gleiche sein muss wie zu ihrer Evaluierung. Dies ist in der linearisierten Bellman-Gleichung zu erkennen:

$$z(i) = \exp(-q(i)) \sum_j \bar{p}_{ij} z(j) = \exp(-q(i)) \mathbb{E}_{j \sim \bar{P}(\cdot|i)}(z(j))$$

Daraus lässt sich eine Update-Regel für eine stochastische Approximation von z ableiten:

$$\hat{z}(i_k) \leftarrow \hat{z}(i_k) + \alpha_k [\exp(-q(i_k)) \hat{z}(i_{k+1}) - \hat{z}(i_k)]$$

Analog zum Q-Learning beschreibt a_k wieder eine Lernrate (üblicherweise $a_k = \frac{1}{k}$). Dieser Algorithmus generiert Samples $(i_k, q(i_k), i_{k+1})$ der unkontrollierten Markov Kette \bar{P} und nutzt diese für die iterative Annäherung. Diese Samples werden zufällig generiert ohne eine bestimmte Strategie zu beachten. Daher ergibt sich der Name Random Z-Learning, denn man kann das zufällige Erzeugen als Random-Strategie interpretieren.

2.4.2 Verbesserte Episodengenerierung

Im folgenden soll nun eine Möglichkeit zur Verbesserung der Episodengenerierung beschrieben werden, sodass sich der Agent gezielter durch die Welt navigiert. Auch hier ist es die Idee, dass der Agent sich an bereits vorhandenen Annäherungen der State-Value Funktion orientiert. Hier ist es, genau wie beim Q-Learning, möglich verschiedenen Strategie, wie zum Beispiel der ϵ -greedy-Strategie, zu folgen. Es soll nun nur eine einfache *greedy* Strategie betrachtet werden nach der immer die zurzeit optimale erscheinende Aktion $\hat{u}^*(i)$ gewählt wird. Die dadurch generierten Samples haben die Form $P(\hat{u}^*(i))$. Die kontrollierten Übergangswahrscheinlichkeiten lassen sich unter dieser Annahme folgendermaßen darstellen:

$$\begin{aligned} p_{ij}(\hat{u}^*(i)) &= \bar{p}_{ij} \exp \left(-\hat{v}(j) - \log \left(\sum_k \bar{p}_{ik} \exp(-\hat{v}(k)) \right) \right) \\ &= \bar{p}_{ij} \frac{\exp(-\hat{v}(j))}{\sum_k \bar{p}_{ik} \exp(-\hat{v}(k))} \end{aligned} \quad (7)$$

Wird diese Art des *greedy* Z-Learnings verwendet, benötigt man zusätzlich noch Importance Sampling. Importance Sampling ist ein Verfahren zur Erzeugung von Stichproben anhand einer Wahrscheinlichkeitsverteilung. Die Idee dabei ist die Verringerung der Varianz. Ziel ist es bei einer Berechnung des Erwartungswertes einer P -verteilten Zufallsvariable die Samples so nach einer geschickt gewählten Verteilung Q zu wählen und zu gewichten, sodass dessen Erwartungswert gleich dem gesuchten Erwartungswert ist. Es

wird also das ursprüngliche Problem durch ein ähnliches, hoffentlich leichters, Problem ersetzt, bei dem das Ergebnis das Gleiche ist. Wie sehr sich das Problem vereinfacht hängt dabei von dem Problem und der geschickten Wahl der neuen Verteilung ab. Besitzt die neue Verteilung Q eine geringere Varianz als P ist ein schnelleres und damit effizienteres Importance Sampling zu erwarten.

Beim Z-Learning wird der erwartete Reward des nächsten Zustand in Bezug auf \bar{P} geschätzt. Todorov schlägt nun vor, die Samples nach $P(\hat{u}^*(i))$ zu erzeugen. Dies führt dazu, dass Zustände mit einem höher geschätzten Z-Wert öfter ausgewählt werden. Der Effekt dieser Erzeugung ist zweierlei. Zuerst werden niedrigere Werte eher nicht betrachtet, denn diese Werte tragen nicht viel zu der Bellman-Optimalitätsgleichung bei. Der andere Effekt liegt darin, dass wenn niedrige Werte weniger auftauchen, werden höhere Werte mit einer höheren Wahrscheinlichkeit gezogen. Diese Werte haben einen größeren Einfluss und sollten daher die Annäherung beschleunigen.

Für den höchstgeschätzten Zustand $m := \arg \max_{i \in S} \hat{z}(i)$ gilt daher:

$$p_{im}(\hat{u}^*(i)) = \bar{p}_{im} \frac{\hat{z}(m)}{\sum_k \bar{p}_{ik} \hat{z}(k)} \geq \bar{p}_{im}.$$

Der Zustand mit der höchsten Bewertung besitzt also eine Wahrscheinlichkeit, die mindestens so hoch ist wie die der unkontrollierten Dynamik \bar{P} . Die Samples werden dann gewichtet mit

$$\frac{\bar{p}_{ij}}{p_{ij}(\hat{u}^*(i))}$$

, sodass gilt

$$\mathbb{E}_{j \sim P(\hat{u}^*(i))[\cdot|i]} \left(\frac{\bar{p}_{ij}}{p_{ij}(\hat{u}^*(i))} z(j) \right) \mathbb{E}_{j \sim \bar{P}(\cdot|i)} (z(j)).$$

Daraus folgt für das einfache *greedy* Z-Learning folgende Update-Regel:

$$\hat{z}(i_k) \leftarrow \hat{z}(i_k) + \alpha_k \left[\frac{\bar{p}_{i_k i_{k+1}}}{p_{i_k i_{k+1}}(\hat{u}^*(i_k))} \exp(-q(i_k)) \hat{z}(i_{k+1}) - \hat{z}(i_k) \right].$$

Wie bereits erwähnt ist eine geschickte Wahl der Verteilung, welche für die Generierung der Samples verantwortlich ist, wichtig für ein effizientes

Verfahren. Eine elegant konstruierte oder gut gewählte Verteilung verringert die Varianz durch die Wahl eines Wertes der wahrscheinlich mehr Einfluss hat. Eine schlecht gewählte Proxy-Verteilung kann sogar einen negativen Effekt haben, sodass das Verfahren länger braucht als mit der ursprünglichen Verteilung. Wie gut die von Todorov gewählte Verteilung $P(\hat{u}^*(i))$ ist lässt sich schwer bewerten. Dies hängt von sehr vielen Faktoren ab, unter anderem die Werte $\hat{z}(i)$ der bereits angenäherten Werte oder auch der Varianz der unkontrollierten Dynamik \bar{P} . Wie bereits gesagt ist dieses *greedy* Verfahren ähnlich des bereits gezeigten *greedy* Q-Learning und daher ist es eine wohl begründete Vermutung, dass auch hier Verbesserung gegenüber der zufälligen Episodengenerierung zu erwarten sind. In [7] wird empirisch gezeigt, dass diese Form des Z-Learnings schneller lernen kann. Ein Problem was dabei jedoch beachtet werden sollte ist, dass die Gewichte Modellinformationen (die Übergangswahrscheinlichkeiten \bar{p}_{ij}) benötigen. Im Allgemeinen und in unseren Annahmen liegt dies aber nicht vor. Eine Lösung für dieses Problem wäre das parallele Lernen dieser Übergangswahrscheinlichkeiten. Dieses Problem wird unter anderem im folgenden Kapitel behandelt, indem versucht wird Z-Learning Aktionen lernen zu lassen auf diskreten Markov Decision Problems.

2.5 Z-Learning lernt Aktionen

Das Z-Learning dem Q-Learning überlegen ist hat mehrere Gründe und kann hier [3] nachgelesen werden. Einer der größten Vorteile die das Z-Learning gegenüber dem Q-Learning hat ist das es sich nur für den Zustandsraum interessiert und nicht wie das Q-Learning den Aktions-Zustandsraum. Das führt dazu, dass es sehr viel schneller lernt, jedoch nicht direkt ein Ergebniss liefern kann, denn die Aktionen um von einen Zustand in den anderen zu kommen sind bekannt. In den meisten Fällen möchte man jedoch genau diese Abbildung von Zustands \rightarrow Aktion a_s wissen. Eine triviale, jedoch sehr oft nicht mögliche, Lösung ist es dem Agenten das Übergangsmodell zu geben. Eine andere Option, welche im Weiteren betrachtet werden soll, ist es dem Agenten dieses Übergangsmodell lernen zu lassen. Im Idealfall während der Agent die State-Value Funktion lernt, somit sollte keine Zeit für zusätzliche Lerniteratoren verloren gehen. Um die Qualität dieses Lernen einschätzen zu können wird das Ergebniss mit dem des Q-Learning verglichen. Ein Problem was sich hierbei jedoch ergibt ist, dass Q- und Z-Learning auf verschiedenen Modellen arbeiten - zum einem diskrete Markov Decision Problems, zum

anderem stetige Markov Decision Problems. Daher wird als erstes versucht werden Z-Learning auf diskrete Markov Decision Problems anzuwenden.

2.5.1 Z-Learning auf diskreten Markov Decision Problems

Beim Versuch Z-Learning auf diskrete Markov Decision Problems anzuwenden ergeben sich folgende Probleme:

1. Z-Learning nutzt eine unkontrollierte Dynamik \bar{P} , die in solcher Form nicht in einem diskreten Markov Decision Problem vorkommt.
2. Beim Z-Learning werden Aktionen als reelle Vektoren dargestellt, welche die unkontrollierte Dynamik \bar{P} beeinflussen. Es gilt daher diese Aktionen auf diskrete Aktionen abzubilden.

Es gibt noch ein weiteres Problem. Möchte man ein verbessertes Z-Learning, zum Beispiel *greedy* Z-Learning, verwenden benötigt man Wissen über die unkontrollierte Dynamik \bar{P} . Hierfür wird Modellwissen benötigt. Dieses Problem zu umgehen ist jedoch Thema dieser Arbeit und wird später untersucht. Zuerst werden Lösungsmöglichkeiten für die ersten zwei Probleme vorgestellt.

2.5.2 Überführung der unkontrollierten Dynamik \bar{P} in eine Strategie

Wie bereits erwähnt gibt es im diskreten Fall keine unkontrollierte Dynamik \bar{P} , sodass zur Episodengenerierung eine einfache Strategie π gewählt wird. Diese Strategie π verteilt die Aktionen gleichmäßig und verwirft Aktionen, die in einem Zustand nicht möglich sind. Eine einfache Überführung dieser unkontrollierten Dynamik \bar{P} in eine Strategie sieht folgendermaßen aus:

$$\forall s \in S : \pi(s, a) = \begin{cases} \frac{1}{|A_s|} & , \text{ falls } a \in A_s \\ 0 & , \text{ falls } a \notin A_s \end{cases}$$

2.5.3 Aktionen als reelle Vektoren in diskrete Aktionen übersetzen

Nach der Beendigung Agent des Lernen der State-Value Funktion müssen die Aktionen, die als reelle Vektoren repräsentiert werden, in diskrete symbolische Aktionen überführt werden. Da der Agent das Übergangsmodell $P_{ss'}^a$, welches er gelernt hat, kennt, sollte der Agent in der Lage sein eine Strategie

zu erstellen, die für jeden Zustand die Aktion wählt, welche die höchste Wahrscheinlichkeit hat im bestbewerteten Nachfolgezustand zu landen. Der beste Nachfolgezustand ist also $P(s'|s) = \sum_{a \in A_s} P_{ss'}^a$. Die Strategie die der Agent am Ende ausgeben sollte hat dann diese Form:

$$\pi(s) = \arg \max_{a \in A_s} (P_{sm}^a), \text{ mit } m = \arg \min_{\{k \in S | P(k|s) > 0\}} (V(k))$$

Somit wird in jedem Zustand die Aktion gewählt die wahrscheinlich im besten Nachfolgezustand landet.

3 Lernen des Übergangsmodell

Zum Lernen des Übergangsmodells werden zwei Ansätze vorgestellt. Zuerst das Lernen von Aktion von jedem Zustand und danach das Lernen von Aktionen von Zustandsgruppen.

3.1 Lernen von Aktionen in jedem Zustand

Dieser Ansatz versucht ein eigenes Übergangsmodell $P'_{ss'}$ zu erstellen, analog zu dem Übergangsmodell $P_{ss'}^a$. Dafür merkt sich der Agent wie oft er bei der Wahl der Aktion a im Zustand s im Nachfolgezustand s' gelandet ist. Der Agent zählt wie oft eine Aktion ihn in welchen Nachfolgezustand bringt. Bei der Auswertung, nach Beendigung des Lernprozess, ist die Wahrscheinlichkeit einer Aktion a im Zustand s' von Zustand s :

$$P'_{ss'}^a = \frac{\text{wie oft in Zustand } s' \text{ gelandet mit Aktion } a \text{ in Zustand } s}{\text{wie oft Aktion } a \text{ in Zustand } s \text{ ausgeführt wurde}}.$$

Diese Darstellung hat den Vorteil, dass sie sofort als Übergangsmodell übernommen werden kann. Der Nachteil liegt in der Größe des Übergangsmodells. Sind sehr viel Zustände oder Aktionen in einem Zustand möglich, ist die Wahrscheinlichkeit hoch, dass eine Aktion in einem Zustand nicht oft genug ausgeführt wurde um repräsentativ für diesen Zustand zu sein. Der Agent könnte zu wenig gelernt haben und hat falsche Vorstellungen von dem wahren Übergangsmodell. Es kann sogar passieren, dass er eine Aktion in einem Zustand nie ausführt und von einer Gleichverteilung ausgehen muss.

Dieses Problem wird dadurch abgeschwächt, dass wenn der Agent mit einer bestimmten Auswahl-Strategie (zum Beispiel ϵ -greedy Strategie) lernt er oft die gleichen Zustände besucht und somit eine größere Stichprobe auf diese erhält.

3.2 Lernen von Aktionen von Zustandsgruppen

Dieser Ansatz geht davon aus, dass der Agent Ähnlichkeiten zwischen Zustände erkennen kann und sich diese ähnlichen Zustände auch ähnlich verhalten. Die Einträge des Übergangsmodell verhalten sich in ähnlichen Zustände also immer gleich. Der Agent zählt hierfür diese Zustandsgruppen. Befindet sich ein Agent in einem Zustand der zu einer Gruppe f von Zuständen gehört und führt die Aktion a aus wird für diese Gruppe gezählt, wobei $f \in F$ und F

sei die Menge aller, dem Agent unterscheidbaren, Zustandsgruppen.

Wenn bei der Auswertung das Übergangsmodell gebildet werden soll muss für alle Zustände erkannt werden welcher Gruppe sie angehört. Ist dies geschehen ist ihre Wahrscheinlichkeitsverteilung analog zu dem Fall des Lernen von jedem Zustand. Es muss jedoch beachtet werden, dass sich die Einträge in der Zeile entsprechend dem Zustand s verschieben, ansonsten würde der Agent davon ausgehen, dass er in jedem Zustand dieser Zustandsgruppe bei Aktion a immer im gleichen Zustand landet.

Es werden also nicht mehr Aktionen in einem bestimmten Zustand betrachtet, sondern Aktionen in einem bestimmten Zustandstyp. Dieser Ansatz hat zwei Nachteile. Es muss zwei zusätzliche Annahme getroffen werden, der Agent kann Ähnlichkeiten erkennen und es gibt Ähnlichkeiten zwischen Zuständen. Weiterhin ist dieser Ansatz eine Verallgemeinerung und kann zu Ungenauigkeiten führen. So können zwei Felder hinreichend ähnlich sein, müssen sich jedoch nicht gleich verhalten - nur ähnlich genug.

Wie vorteilhaft diese Ansätze für das Z-Learning sind wird betrachtet, nachdem das Setting, auf dem gearbeitet wird, erklärt wurde.

4 Setting

Um zu zeigen wie sich diese Ideen auswirken begeben wir uns in ein spezifisches Szenario. Als Szenario wählen wir die Gridworld.

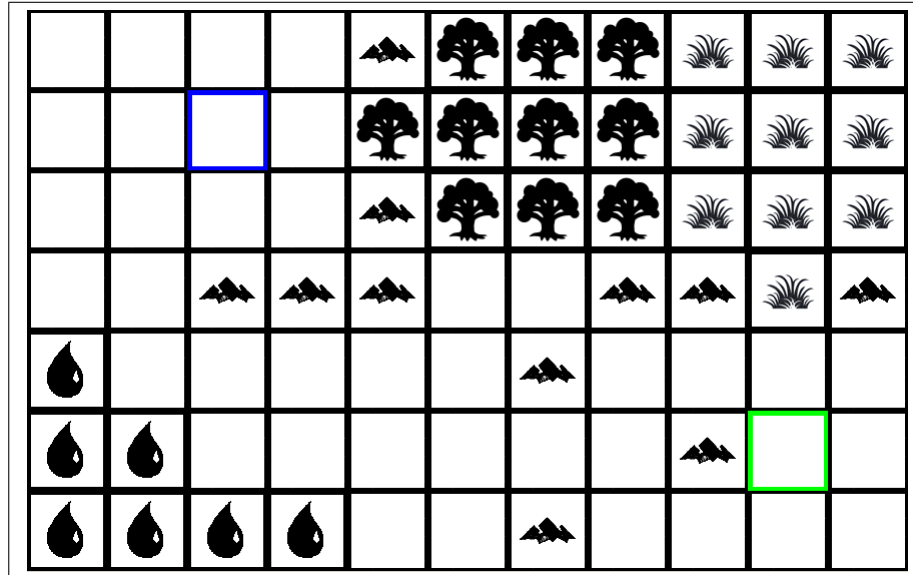


Abbildung 1: Gridworld Beispiel

In Abbildung 1 sieht man eine Beispielwelt mit allen Elementen. Jedes Feld ist ein Zustand. Die Zustandsmenge ist abhängig von der Breite n und der Höhe m der Gridworld, sodass:

$$S = \{1, \dots, n\} \times \{1, \dots, m\}.$$

Der Agent hat auf jedem Feld vier Aktionen zur Verfügung:

$$\forall s \in S : A_s = \{\leftarrow, \rightarrow, \uparrow, \downarrow\}.$$

Das Ziel des Agenten ist es vom Startfeld zum Endfeld zu gelangen. Das Startfeld hat einen blauen Rand, während alle Zielfelder einen grünen Rand haben. Zielfelder sind absorbierend. Erreicht der Agent ein Ziel gilt die Episode als beendet und es wird eine neue Episode gestartet, wobei der Agent wieder auf dem Startfeld ist.

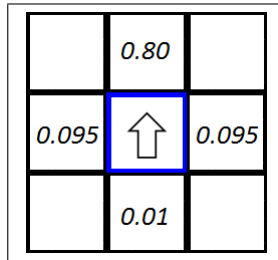


Abbildung 2: Aktionsschema einer Ebene

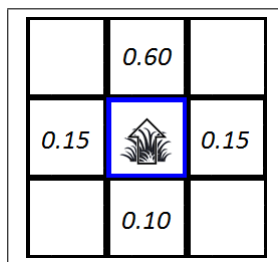


Abbildung 3: Aktionsschema eines Sumpfes

Es gibt mehrere Feldtypen. Diese verhalten sich immer gleich und der Agent ist in der Lage zu erkennen auf was für einem Feld er sich befindet. Als Beispiel für das Aktionsschema soll Abbildung 2 dienen. Der Agent befindet sich dabei auf dem Startpunkt. Er führt die Aktion \uparrow aus und die Zahlen geben an auf welchem Feld er mit welcher Wahrscheinlichkeit landet. Analog für die anderen Aktionen entsprechend ihrer Richtung gedreht. Im Gegensatz dazu sind in Abbildung 3 die Übergangswahrscheinlichkeiten anders, da der Agent sich auf einem Sumpf-Feld befindet und dieser schwieriger zu durchqueren ist.

Befindet sich der Agent neben einem nicht betretbarem Feld oder dem Rand und versucht dieses Feld zu betreten oder die Gridworld zu verlassen, bleibt er stattdessen auf seinem Feld stehen.

Die Kosten sind abhängig von jeweiligem Feldtypen. Einzige Ausnahme hier bilden die Ziele welche immer Kosten von 0 haben um absorbierende Zustände zu schaffen. Die Lernrate α aller Agenten ist immer skaliert auf die Anzahl der Bewegungen c des Agenten in der aktuellen Episode.

$$\alpha_c = \frac{1}{c}$$

Auf der folgenden Seite befindet sich eine Tabelle mit allen Feldtypen des Szenarios. Die Spalten „Bild“, „Beschreibung“ und „Kosten“ sollten selbsterklärend sein. Die nächsten drei Spalten beschreiben das Übergangsmodell. Wählt ein Agent die Aktion \uparrow und landet im nächsten Schritt auch auf dem Feld über ihm so gilt dies als Erfolg und die Wahrscheinlichkeit dafür ist in dieser Spalte beschrieben. Landet er einem Feld rechts oder links von ihm ist dies die Wahrscheinlichkeit in der „90° Drehung“ Spalte. Hierbei ist zu beachten, dass nur eine Wahrscheinlichkeit aufgelistet ist - entweder nach rechts oder nach links. Die letzte Spalte beschreibt den Fall, dass der Agent in die vollkommen falsche Richtung geht. In diesem Fall wäre es nach unten gehen. Mit diesen Informationen haben wir ein diskretes MDP beschrieben und wir können untersuchen wie sich Z-Learning im Vergleich zu Q-Learning verhält, wenn es versucht die Umgebung zu lernen.






Feld	Beschreibung	Kosten	Erfolg	90° Drehung	180° Drehung
	Ebene. Das Basisfeld, die Kosten für das Betreten aller anderen Felder sind auf dieses Feld normiert.	1	0.80	0.095	0.01
	Berg. Dieses Feld ist ein Blocker und kann vom Agenten nicht betreten werden	—	—	—	—
	Wald. Dieses Feld ist etwas schwieriger zu passieren und der Natur nach verwirrender.	1.2	0.60	0.15	0.10
	Sumpf. Der Sumpf ist ähnlich zum Wald. Er ist verwirrend und noch schwieriger zu durchqueren.	2	0.60	0.15	0.10
	Wasser. Gleich dem Berg. Es ist nur aus ästhetischen Gründen vorhanden.	—	—	—	—

Tabelle 1: Feldtypen der Gridworld

4.1 Ergebnisse

Im Folgenden werden die Ergebnisse der Tests der verschiedenen Ansätze und Algorithmen auf verschiedenen Gridworlds dargestellt. Zuerst wird untersucht, ohne Q- oder Z-Learning, wie gut sich das Übergangsmodell lernen lässt. Danach nutzt das Z-Learning verschiedene Lernmethoden um das Übergangsmodell zu verstehen und vergleicht seine Ergebnisse mit denen des Q-Learnings.

4.1.1 Fehlerbeschreibung

Zum Vergleich von Strategien wird der normierte, mittlere Approximationsfehler je Episode verwendet. P_{ss}^a beschreibt dabei das errechnete Modell, während $P_{ss'}^a$ das im Hintergrund liegende Modell ist. Der Fehler auf einem Feld ist dann:

$$\sigma_{xyd} = |P_{xy}^d - P_{xy}^a|.$$

Der Fehler insgesamt ist dann das Mittel der Einzelfehler:

$$\sigma = \frac{1}{4nm} \sum_{i=1}^m \sum_{j=1}^n \sum_{k \in \{\leftarrow, \rightarrow, \uparrow, \downarrow\}} \sigma_{ijk}$$

Weiterhin ist anzumerken, dass jeder Graph sowohl auf der Abzisse als auch der Ordinate logarithmische Skalierung aufweist.

4.1.2 Lernen von Aktionen in jedem Zustand

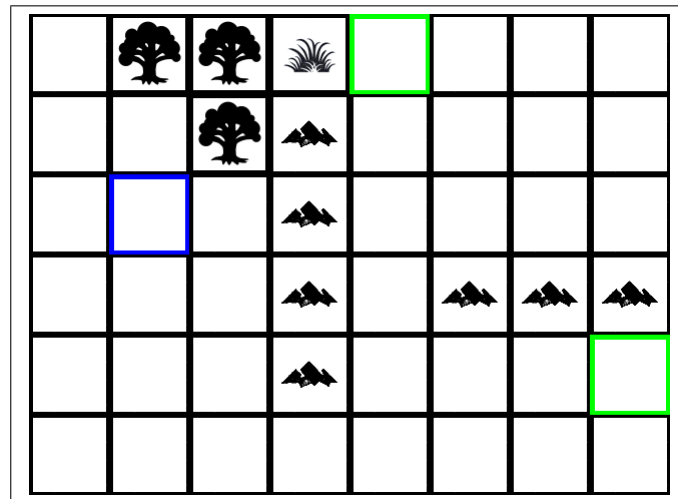


Abbildung 4: Mittlere Testwelt

Getestet wird auf der Welt in Abbildung 4. Sie ist nicht so groß und kann daher recht schnell erkundet werden, dabei bleibt sie durch mehrere Ziele, Feldtypen und Wege zum Ziel interessant. Der Agent den wir testen bewegt sich zufällig durch die Welt und hat keine Ambitionen das Ziel zu finden.

Gelernt werden die Aktionen in jedem Zustand. Der Agent achtet also nicht auf den Feldtypen.

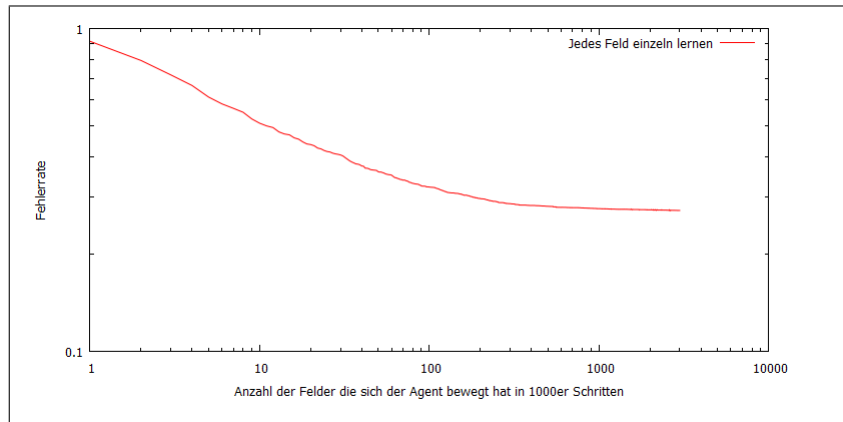


Abbildung 5: Entwicklung des Fehlers auf mittlerer Testwelt beim Lernen jedes Felds

In Abbildung 5 erkennt man, dass das Lernen der Felder einzeln nicht effektiv ist. Nach 250.000 Bewegungen stabilisiert sich die Fehlerrate auf circa 28% und verbessert sich danach nicht signifikant. Dies ist dadurch zu erklären, dass der Agent wegen der schieren Größe nicht alle möglichen Wege oft genug ausprobieren kann um eine repräsentative Darstellung des Übergangsmodell zu erhalten. Zum Beispiel ist auf dem Feld (6, 2) der Agent niemals nach Norden gegangen und ist auf dem Feld östlich von ihm gelandet. Dies lässt sich nicht aus dem Bild oder dem Graphen erkennen, sondern ist ein beispielhafter Auszug aus den Testdaten. Vergrößern wir die Welt sogar weiter (Abbildung 6) wird der Fehler noch größer. Der Fehler stabilisiert sich auf 48% nach circa 100.000 Bewegungen. Dieser Ansatz eignet sich also maximal für sehr kleine Szenarien.

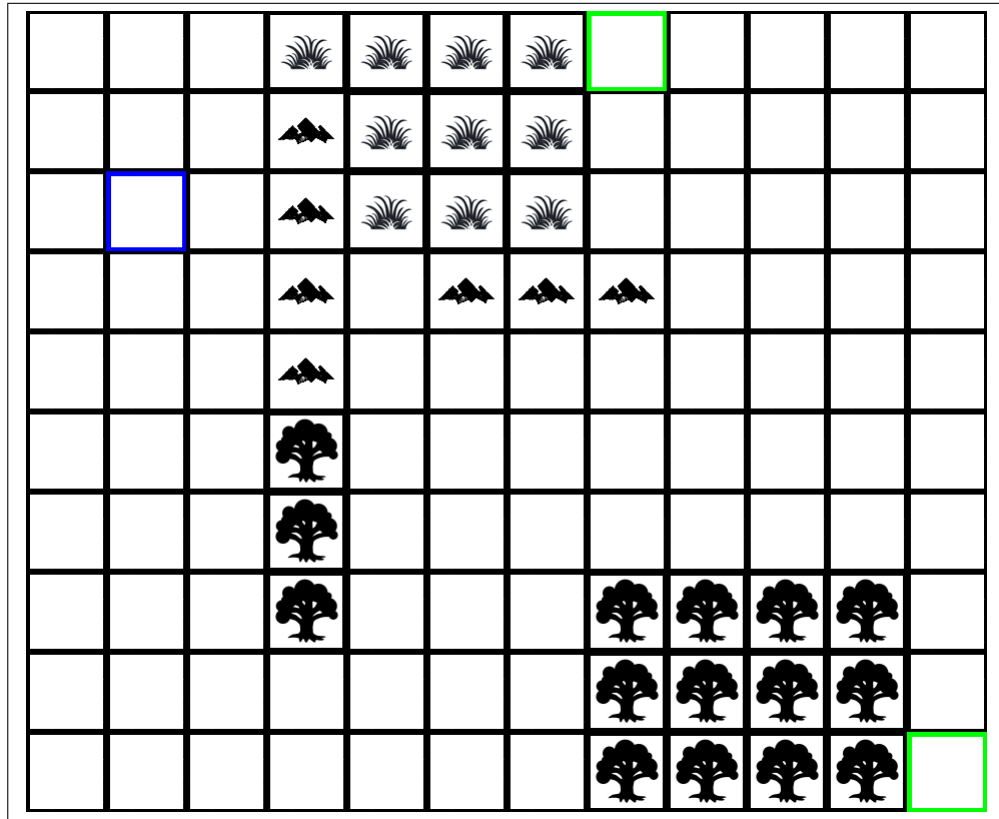


Abbildung 6: Große Testwelt

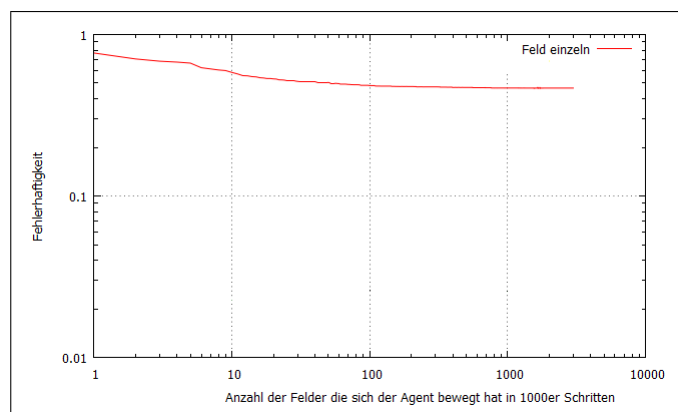


Abbildung 7: Entwicklung des Fehlers auf großer Testwelt beim Lernen jedes Felds

4.1.3 Lernen von Aktionen von Zustandsgruppen

Hier wird auf den gleichen Testwelten getestet wie im Versuch jedes Feld einzeln zu lernen. Es existiert eine mittlere und eine große Testwelt.

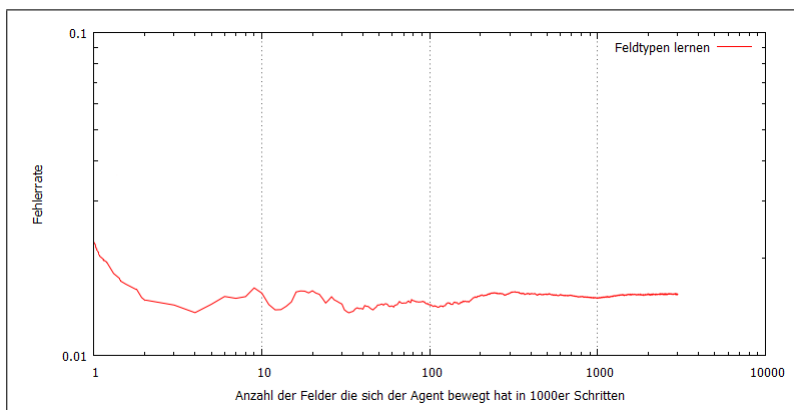


Abbildung 8: Entwicklung der Fehlerrate auf großer Welt beim Lernen von Feldtypen

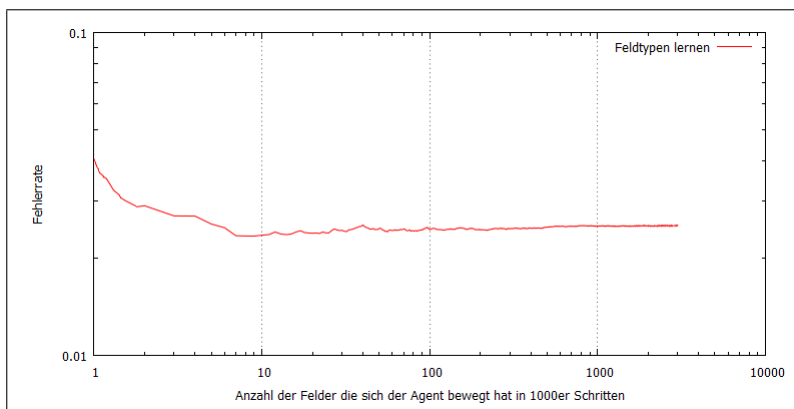


Abbildung 9: Entwicklung der Fehlerrate auf mittlerer Welt beim Lernen von Feldtypen

Den Effekt erkennt man auch sofort beim Vergleich der Graphen 8 und 9. Die mittlere Welt hat einen größeren Fehler als die große Welt, da es dort mehr Ränder gibt. Sowohl in der großen als auch in der mittleren Welt stabilisieren sich die Fehlerrate sehr schnell. Die große Welt braucht 200.000 Bewegungen und die normale 100.000, etwas weniger als der Agent der jedes Feld einzeln lernt. Um weiten besser sind jedoch die Fehlerraten 1,5% für die große Welt und 2,4% für die normale Welt. Innerhalb der ersten 50.000 schwanken die Fehlerraten noch stark, der Grund dafür ist die Größe der Stichproben.

4.1.4 Vergleich Q- und Z-Learning

Um Vergleichen zu können, wie gut oder schlecht sich das Lernen der Aktionen auswirkt vergleichen wir zuerst Q- mit Z-Learning. Beide folgen zur Auswahl der nächsten Aktion einer *greedy* Strategie. Das Z-Learning nutzt Importance-Sampling. Dem Z-Learning ist zu jedem Zeitpunkt das komplette Übergangsmodell bekannt. Getestet wird hier nur auf der mittleren Testwelt.

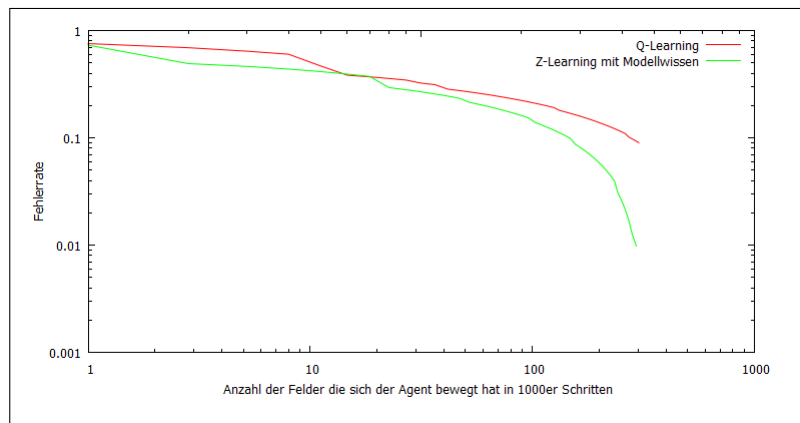


Abbildung 10: Entwicklung der Fehlerrate auf mittlerer Welt von Q- und Z-Learning. Z-Learning besitzt das komplette Modell und lernt nicht die Aktionen

Abbildung 10 zeigt den Verlauf der beiden Algorithmen auf der mittleren Welt. Hier erkennt man auch deutlich die Überlegenheit des Z-Learnings gegenüber dem Q-Learning. Es vermindert seinen Fehler schneller und stärkerem Maße. Aufgrund der logarithmischen Skalierung sieht es so aus, als ob es erst in der zweiten Hälfte die Algorithmen ihre Fehler verbessern. Dies trägt jedoch, so hat der Z-Learning Agent seinen Fehler nach bereits 17000 Schritten (nicht Episoden!) auf unter 20% gebracht, während das Q-Learning noch bei 25% ist. In den Folgenden Schritten wird der Fehler immernoch minimiert, jedoch nie besser als 1% für das *greedy* Z-Learning und 3% für das *greedy* Q-Learning .

Als nächstes wird das *greedy* Z-Learning ersetzt mit einem Random Z-Learning, der die Aktionen von jedem Feld einzeln lernt.

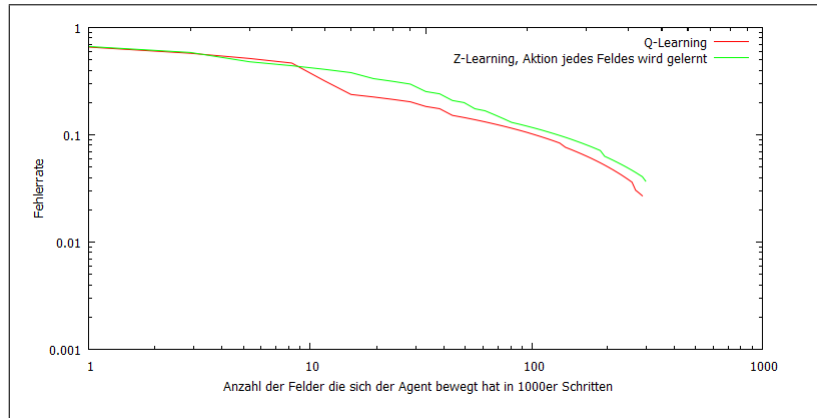


Abbildung 11: Entwicklung der Fehlerrate auf mittlerer Welt von Q- und Z-Learning. Z-Learning lernt die Aktionen für jedes Feld einzeln.

In Abbildung 11 sieht man den Effekt. Das Z-Learning ist kurzzeitig besser als das Q-Learning, aber auch nur weil das Q-Learning sich in den nächsten Schritten hervorragend verbessert. Danach ist das Z-Learning immer etwas schlechter als das Q-Learning. Dies liegt daran, dass die Aktionen nicht erfolgreich gelernt wurden. Der Fehler lag für das Lernen der Aktionen der einzelnen Felder auch später noch weit über 25% und somit ist keine sinnvolle Transformation der State-Value Funktion in eine Action-Value Funktion zu erwarten.

Als letztes wird das Random Z-Learning betrachtet, bei dem die Aktionen je Feldtyp gelernt werden. Eine Verbesserung gegenüber dem Lernen der Aktionen der einzelnen Felder ist zu erwarten. Spannend ist jedoch die Frage wie gut sie im Vergleich zum *greedy* Z-Learning mit Importance-Sampling ist.

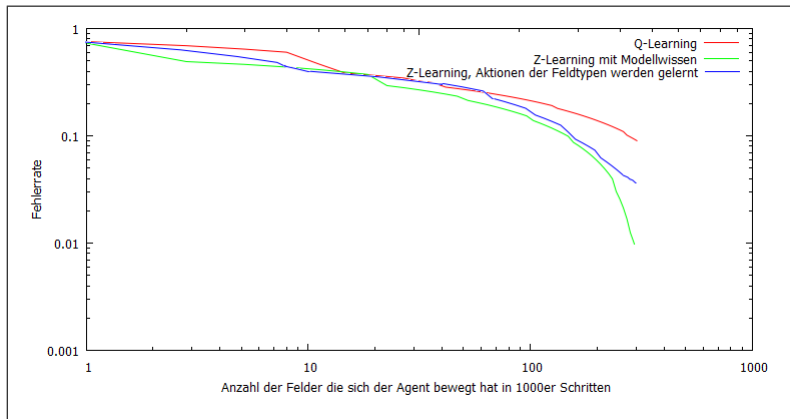


Abbildung 12: Entwicklung der Fehlerrate auf mittlerer Welt von Q- und Z-Learning. Z-Learning lernt die Aktionen für jedes Feld einzeln.

In Abbildung 12 werden die Erwartungen bestätigt. Es ist eine deutliche Verbesserung zum Lernen Aktionen der einzelnen Felder zu sehen, denn es ist besser als das Q-Learning. Dass es nicht genausogut wie das Z-Learning ist war zu erwarten, da letztendlich immer ein Fehler bestehen wird zwischen dem errechneten Modell und dem echten Modell. Besonders am Anfang ist dies zu sehen, der Agent hält sich nahe an der Q-Learning Kurve. Ein besonderer Erfolg ist die Tatsache, dass diese Art des Z-Learnings sich insgesamt besser schlägt als das Q-Learning. Nicht nur im Bereich des Endergebnis, sondern auch am Anfang. Im mittleren Teil ist das Z-Learning jedoch kurzzeitig schlechter, verbessert sich dann sehr stark, während das Q-Learning sich nur leicht verbessert.

5 Zusammenfassung

5.1 Auswertung

Nachdem verschiedene Ansätze zur Entfernung der Annahme, dass das Z-Learning ein Übergangsmodell benötigt, in verschiedenen Experimenten untersucht und unter einander verglichen wurde, lässt sich eine Antwort auf diese Frage geben. Die Ergebnisse belegen, dass Z-Learning kein Modell benötigt um zu funktionieren. In allen Fällen konvergiert das Z-Learning ohne Modell gegen eine optimale Strategie. Außerdem ist es dabei mit Einschränkungen besser als der klassische Ansatz des Q-Learnings auf diskreten Markov Decision Problems.

Die Nachteile gegenüber dem normalen Z-Learning sind jedoch Viele. Der neue Algorithmus braucht mehr Speicherplatz um sich die Anzahl der gemachten Aktionen zu merken, jedoch noch lange nicht soviel wie das Q-Learning, welches mindestens $|S| \times |A|$ Speicherplatz benötigt. Weiterhin wurde es geschafft die Annahme des Vorhandensein des Übergangsmodell zu entfernen, dafür wurde die Annahme aufgestellt, dass die Zustände in verschiedene Typen kategorierbar sind und der Agent diese erkennen kann. Diese Annahme ist sehr viel schwächer. Möchte man auch ohne diese auskommen, ist ein Wechsel zum Q-Learning angebracht, denn das Q-Learning ist dem Ansatz, des Lernen aller Aktionen in jedem Zustand, überlegen.

5.1.1 Ausblick

Diese Ergebnisse geben Anlass zur weiteren Betrachtung und Analyse des Z-Learnings. Thema weiterer Untersuchung könnten interessantere Gridworlds sein. Könnten sich zum Beispiel die Felder über die Zeit verändern (Gewitter oder Schnee), wodurch sich die Kosten und das Übergangsmodell verändern für dieses Feld. Es könnten auch andere Zustandsräume erkundet werden, die nicht eine Gridworld darstellen, so könnte man versuchen einen Agenten ein Spiel wie Go oder Schach lernen zu lassen. Bei diesen Beispielen wäre es besonders interessant sich mit der Frage zu beschäftigen wie die einzelnen Verfahren sich bei größeren Zustands- und Aktionsmengen verhalten. Es ist zu erwarten, dass das Q-Learning noch weiter gegenüber dem Z-Learning mit Modellwissen zurückfällt. Doch hier ist eine Modellierung der Ähnlichkeit, in Bezug auf Z-Learning mit Lernen der Zustandstypen, interessant. Bei diesen Spielen gibt es nie wirklich mehrmal den exakt gleichen

Zustand, sodass Ähnlichkeit Domänenwissen voraussetzt oder eine clevere Ähnlichkeitsdefinition.

5.1.2 Schluss

In dieser Arbeit wurden die Grundzüge von diskreten und stetigen Markov Decision Problems erläutert, was Reinforcement Learning ist und die Grundlagen des Q- und Z-Learning beschrieben. Zudem wurden Lösungen beschrieben und untersucht, wie man Z-Learning ohne Modell ausführen kann. Die Ergebnisse wurden verglichen mit den Ergebnissen von etablierten Verfahren und die Tauglichkeit von Z-Learning ohne Modell gezeigt.

Literatur

- [1] Richard E. Bellman. *Dynamic Programming. Dover paperback (2003)*. Princeton University Press, 1957.
- [2] Stuart E. Bellman, Richard E. ; Dreyfus. *Applied Dynamic Programming*. Princeton University Press, 1962.
- [3] Vincent Froese. *Vergleich von Z-Learning und Q-Learning auf diskreten Markov Decision Problems*. Technische Universität Berlin, 2010.
- [4] Tommi Jaakkola, Michael I. Jordan, and Satinder P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 1994.
- [5] Stuart Russel. *Künstliche Intelligenz - Ein moderner Ansatz*. Pearson Studium, 2004.
- [6] Emanuel Todorov. Linearly-solvable markov decision problems. *Advances in Neural Information Processing Systems*, 2006.
- [7] Emanuel Todorov. Efficient computation of optimal actions. *Proceedings of the National Academy of Sciences of the United States of America*, 2009.
- [8] John N. Tsitsiklis. Asynchronous stochastic approximation and q-learning. *Machine Learning*, 1994.
- [9] Peter Watkins, Christopher J. C. H. ; Dayan. Vergleich von z-learning und q-learning auf diskreten markov decision problems. *Machine Learning*, 1992.

Abbildungsverzeichnis

1	Gridworld Beispiel	29
2	Aktionsschema einer Ebene	30
3	Aktionsschema eines Sumpfes	30
4	Mittlere Testwelt	34
5	Entwicklung des Fehlers auf mittlerer Testwelt beim Lernen jedes Felds	35
6	Große Testwelt	36
7	Entwicklung des Fehlers auf großer Testwelt beim Lernen jedes Felds	36
8	Entwicklung der Fehlerrate auf großer Welt beim Lernen von Feldtypen	37
9	Entwicklung der Fehlerrate auf mittlerer Welt beim Lernen von Feldtypen	37
10	Entwicklung der Fehlerrate auf mittlerer Welt von Q- und Z- Learning. Z-Learning besitzt das komplette Modell und lernt nicht die Aktionen	39
11	Entwicklung der Fehlerrate auf mittlerer Welt von Q- und Z- Learning. Z-Learning lernt die Aktionen für jedes Feld einzeln.	40
12	Entwicklung der Fehlerrate auf mittlerer Welt von Q- und Z- Learning. Z-Learning lernt die Aktionen für jedes Feld einzeln.	41